arXiv:1901.00069v1 [cs.LG] 1 Jan 2019

Recurrent Neural Networks for Time Series Forecasting

Gábor Petneházi*

Doctoral School of Mathematical and Computational Sciences
University of Debrecen

Abstract

Time series forecasting is difficult. It is difficult even for recurrent neural networks with their inherent ability to learn sequentiality. This article presents a recurrent neural network based time series forecasting framework covering feature engineering, feature importances, point and interval predictions, and forecast evaluation. The description of the method is followed by an empirical study using both LSTM and GRU networks.

1 Introduction

Recurrent neural networks are well suited to supervised learning problems where the dataset has a sequential nature. Time series forecasting should not be an exception.
RNNs are essentially neural networks with memory. They can remember things from the past, which is obviously useful for predicting time-dependent targets. Yet, applying them to time series forecasting is not a trivial task. The aim of this article is to review some practices that can help RNNs deliver useful forecasts.

2 Neural networks for forecasting

The objective of supervised learning is to predict something from data. A training set of an output (target) and some input variables is fed to an algorithm that learns to predict the target values. The output may be categorical (classification) or continuous (regression). The task of the algorithm is to deliver high quality predictions, all by itself, extracting the required knowledge solely from the available data.
Neural networks are a popular framework for supervised learning — a network-like system of weighted summations and differentiable functions that can learn astoundingly complex things. Usually, variants of gradient descent together with backpropagation (chain rule) are used to find the optimal values of the network weights. This is all simple and intuitive, yet the resulting networks are usually

*gabor.petnehazi@science.unideb.hu

difficult for people to understand. There are so many weights and connections that we just wonder how the system produced the results. We don't understand neural networks, but we like them. The reason for their ever high popularity is simple: they are good. They can learn arbitrarily complex functions, and they often provide excellent predictions for pretty difficult machine learning problems.

NNs are widely used in machine learning, time series prediction is just one example application. Werbos [1974] and Werbos [1988] made a pioneering work in the field of neural networks by developing a general formulation of backpropagation. Werbos applied the method to forecasting, and compared it to traditional forecasting methods. Tang et al. [1991] also made a neural networks vs. Box-Jenkins comparison and found that NNs outperform the Box-Jenkins model for series with short memory. For series with long memory, both methods produced similar results. Faraway and Chatfield [1998] compared neural networks with Box-Jenkins and Holt-Winters methods for forecasting and found that the design of network architecture and the choice of input variables require great care, and so applying neural networks in black box mode is not a good idea. They also found that increasing the number of hidden nodes may deteriorate out-of-sample performance.

Zhang and Qi [2005] found that neural networks are not able to capture seasonality by default, and deseasonalization and detrending can help their forecasting performance. According to Balkin and Ord [2000], differencing is unnecessary for neural network based forecasting, but a log transformation may be beneficial.

Zhang et al. [1998] gives a detailed review of neural networks for forecasting. Gamboa [2017] provides a more recent review of the applications of deep learning to time series data.

3 Recurrent neural networks

RNNs are neural networks for sequential data — hereby we apply them to time series. The main idea behind recurrent neural networks is using not only the input data, but also the previous outputs for making the current prediction. This idea makes a lot sense — we could build neural networks passing values forward in time. However, such simple solutions usually do not work as expected. They are hard to train and they are forgetful. Rather, we need to have a system with some kind of memory.

There are two popular and efficient RNN models that work really well: long short-term memory and gated recurrent unit.

3.1 LSTM

Long short-term memory (Hochreiter and Schmidhuber [1997]) is a gated memory unit for neural networks. It has 3 gates that manage the contents of the memory. These gates are simple logistic functions of weighted sums, where the weights might be learnt by backpropagation. It means that, even though it seems a bit complicated, the LSTM perfectly fits into the neural network and its training process. It can learn what it needs to learn, remember what it needs to remember, and recall what it needs to recall, without any special training

or optimization. The input gate (1) and the forget gate (2) manage the cell state (4), which is the long-term memory. The output gate (3) produces the output vector or hidden state (5), which is the memory focused for use. This memory system enables the network to remember for a long time, which was badly missing from vanilla recurrent neural networks.

$$i_t = sigmoid\,(W_i x_t + U_i h_{t-1} + b_i) \tag{1}$$

$$f_t = sigmoid\,(W_f x_t + U_f h_{t-1} + b_f) \tag{2}$$

$$o_t = sigmoid\,(W_o x_t + U_o h_{t-1} + b_o) \tag{3}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh\,(W_c x_t + U_c h_{t-1} + b_c) \tag{4}$$

$$h_t = o_t \odot tanh\,(c_t) \tag{5}$$

3.2 GRU

Gated recurrent unit (Cho et al. [2014]) is essentially a simplified LSTM. It has the exact same role in the network. The main difference is in the number of gates and weights — GRU is somewhat simpler. It has 2 gates. Since it does not have an output gate, there is no control over the memory content. The update gate (6) controls the information flow from the previous activation, and the addition of new information as well (8), while the reset gate (7) is inserted into the candidate activation. Overall, it is pretty similar to LSTM. From these differences alone, it is hard to tell, which one is the better choice for a given problem. For a comparison, see Chung et al. [2014].

$$z_t = sigmoid\,(W_z x_t + U_z h_{t-1} + b_z) \tag{6}$$

$$r_t = sigmoid\,(W_r x_t + U_r h_{t-1} + b_r) \tag{7}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot tanh\,(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{8}$$

3.3 Recurrent neural networks for forecasting

Though it is probably not their primary application, LSTM and GRU networks are often used for time series forecasting. Gers et al. [2002] used LSTMs with peephole connections to learn temporal distances. Malhotra et al. [2015] used stacked LSTM networks to detect anomalies in time series. Guo et al. [2016] proposed an adaptive gradient learning method for RNNs that enables them to make robust predictions for time series with outliers and change points. Hsu [2017] incorporated autoencoder into LSTM to improve its forecasting performance. Cinar et al. [2017] proposed an extended attention mechanism to capture periods and model missing values in time series. Bandara et al. [2017] used LSTMs on groups of similar time series identified by clustering techniques. Laptev et al. [2017] applied RNNs to special event forecasting and found that neural networks might be a better choice than classical time series methods when the number, the length and the correlation of the time series are high. Che et al. [2018] built a GRU-based model with a decay mechanism to capture informative missingness in multivariate time series.
Several attempts have been made on better understanding RNNs. Karpathy et al. [2015] explored the source of recurrent neural networks' performance with

performance comparisons and error analysis on character level language models. van der Westhuizen and Lasenby [2017] used different datasets to visualize the operations of LSTMs. Greff et al. [2017] compared the performance of several LSTM variants, and found that the forget gate and the output activation function are the most important elements of an LSTM block. Chang et al. [2017] proposed a feature ranking method using variational dropout (Hinton et al. [2012]).
Some studies tried to find ways to measure the uncertainty associated with the time series forecasts of recurrent neural networks. Zhu and Laptev [2017] made uncertainty estimates using Monte Carlo dropout and an encoder-decoder framework of LSTM units. Caley et al. [2017] constructed prediction intervals for convolutional LSTMs using bootstrapping.
Here we are going to explore different aspects of RNN-based time series forecasting, and introduce an end-to-end framework for producing meaningful forecasts.

4 Features

4.1 Feature engineering

LSTM and GRU networks can learn and memorize the characteristics of time series. It is not so easy though, especially when we only have a short series of values to learn from. Smart feature engineering can help.
There are very few things whose future values we certainly know. Time is one such thing — we always know how it passes. Therefore, we can use it to make forecasts, even for multiple steps ahead into the future, without increasing uncertainty. All we have to do is extracting useful features that our algorithm can easily interpret.
Time series components, such as trend or seasonality can be encoded into input variables, just like any deterministic event or condition. Time-shifted values of the target variable might also be useful predictors.
Features are usually normalized before being fed to the neural network. It is beneficial for the training process. Two popular choices for rescaling the variables are the minmax scaler (9) and the standard scaler (10).

$$\tilde{x} = \frac{x - min(x)}{max(x) - min(x)} \tag{9}$$

$$\tilde{x} = \frac{x - mean(x)}{sd(x)} \tag{10}$$

4.1.1 Lags

Lagging means going some steps back in time. To predict the future, the past is our best resource — it is not surprising that lagged values of the target variable are quite often used as inputs for forecasting. Lags of any number of time steps might be used. The only drawback of using lagged variables is that we lose the first observations — those whose shifted values are unknown. This might be a matter when the time series is short.

4.1.2 Trend

Trend can be grabbed by features indicating the passage of time. A single variable of equidistant increasing numbers might be enough for that.

4.1.3 Seasonality

We may try to find repetitive patterns in the time series by encoding seasonal variations. There are different ways to do this.
One-hot encoding is a reasonable choice. Hereby, we treat seasonality as categorical variables, and use dummy variables to indicate the current time interval in the seasonal cycle. It is simple and intuitive. However, it can not really grab cyclicality, since the distance between intervals does not matter during the encoding. It means that two successive, and possibly similar, time intervals are represented by just as independent values as any two randomly chosen time intervals. Also, one-hot encoding uses an individual variable to represent each unique value, which may be inconvenient when we have a large number of time intervals. These deficiencies of the dummy variable approach lead us to another encoding method.
We can place the values on a single continuous scale instead of using several binary variables. By assigning increasing equidistant values to the successive time intervals, we can grab the similarity of adjacent pairs, but this encoding sees the first and the last intervals as being farthest from each other, which is a bad mistake. This may be healed by transforming the values using either the sine (11) or the cosine (12) transformation. In order to have each interval uniquely represented, we should use both.

$$\tilde{x} = \sin\left(\frac{2 \cdot \pi \cdot x}{max(x)}\right) \tag{11}$$

$$\tilde{x} = \cos\left(\frac{2 \cdot \pi \cdot x}{max(x)}\right) \tag{12}$$

4.1.4 Dummy indicators

We can use simple indicator variables for events or conditions that we consider important. Holidays are always special and are spent in unusual ways. Hence, a binary variable indicating holidays may carry information about the time series. Also, an indicator of working days or working hours could be useful.

4.2 Feature importances

Neural networks consist of connected simple functions of weighted sums. It is not a difficult structure, yet interpreting the meaning and role of the numerous backpropagation learnt weights is pretty hard. It is the reason for too often calling them black boxes, and not even trying to understand them. Recurrent networks are even a bit more complicated.
This difficulty of interpretation makes neural networks somewhat less valuable. We should naturally always prefer a simpler, more interpretable model, when having multiple choices with about the same forecasting performance. So, we should brighten the black box, at least partially.

A measure of variable importance could tell something about what is happening within the neural network, yet quantifying importance is not trivial. Several methods have been proposed, see Gevrey et al. [2003] or Olden et al. [2004] for a comparison.
Here we are going to use mean decrease accuracy — a method that is usually applied to random forests (Breiman [2001]). It is also called permutation accuracy. We are permuting the variables, one by one, and calculate a measure of accuracy for each deliberately corrupted model. The feature who's random permutation leads to the largest drop in accuracy is considered the most important. It is a simple, intuitive measure, and it can easily be applied not just to random forests, but to any neural network or any other supervised learning algorithm as well.
Two metrics will be used as measures of accuracy: R^2 and MDA. R^2 or coefficient of determination is a goodness of fit measure for regression. It usually ranges from 0 to 1, though it can take on negative values as well. MDA or mean directional accuracy is the accuracy of the binary variable indicating the directional change of consecutive values. Mean decrease accuracy can be calculated from both accuracy measures.
Variables are permuted separately, so the importance scores should be interpreted and evaluated independently. Variable importances do not add up. Hence, when the same piece of information is encoded into multiple variables, we are going to calculate the maximum of the scores, and use it as the importance of that group of variables.
Feature importances can be estimated from both regression accuracy and directional accuracy of one-step (and maybe also multi-step) predictions on different validation sets. It means there are quite some combinations, and it is likely that not the exact same variables will prove to be important in all cases. Therefore, it is worth computing some descriptive statistics of the different calculations to get a summarized view of the actual roles of input features.
The importance scores are usually normalized to sum to 1.

5 Prediction

5.1 Point predictions

Our recurrent neural networks are primarily suited for one-step-ahead forecasting. We have a single output variable containing the value that immediately follows the corresponding sequence of inputs.
This simple framework can also be used for multi-step-ahead forecasting in an iterative (recursive) manner. When the target variable's lagged values are the only non-deterministic features, we may use the predicted values as inputs for making further predictions. It is a bit risky though, since we roll the errors forward.

5.2 Prediction intervals

Neural networks have an important disadvantage: they only provide point estimates. We only get some forecasted values, without any indicators of the predictions' confidence.

Prediction intervals contain future values with a given probability. Such interval estimates could make our forecasts more meaningful, though producing them for neural networks is not easy. We are going to use the computationally expensive method of bootstrapping.

The idea of bootstrapping was introduced by Efron et al. [1979]. This is actually a very simple method for estimating the quality of estimates. Bootstrapping takes resamples with replacement from the original dataset to make separate forecasts. The variability of those independent estimates is a good measure of the confidence of the estimation.

Efron and Tibshirani [1986] wrote a comprehensible review of bootstrapping, focusing on applications. They state that the bootstrap method has very nice properties for estimating standard errors, and computational intensity, being its main weakness, is less and less of a case as computation is getting cheaper and faster. Efron and Tibshirani also note that the bootstrap can be used to construct confidence intervals in an automatic way, while alternative methods require several tricks to perform satisfactorily.

Bootstrapping is not only useful for quantifying the confidence in our predictions, but also for improving the forecasts themselves. Ensemble methods combine multiple algorithms to deliver better predictions. Bootstrapping is one way to construct an ensemble.

There are 2 main approaches to ensemble learning (Dietterich [2002]). Either we can create models in a coordinated fashion, and take a weighted vote of the components, or we can construct independent models with a diverse set of results, then combine them to let the disagreements cancel out. Bootstrap resampling can be used to construct an ensemble of this latter type.

Bagging or bootstrap aggregating is an ensemble method that trains a learning algorithm on multiple independent bootstrap samples, and aggregates the resulting predictors by voting or averaging. It was introduced by Breiman [1996]. This simple ensemble can improve unstable single predictors, but it can slightly degrade the performance of more stable ones. Hence, applying bagging is not always a good idea, but it can work very well for some learning algorithms. Dietterich [2002] remarks that since the generalization ability of neural networks is very good, they may benefit less from ensemble methods. Still, we may hope that bagging can bring some improvement to our point predictions, but the main reason for applying the method is the construction of prediction intervals.

Bootstrapping has been applied to compute confidence and prediction intervals for neural networks by, e.g., Paass [1993], Heskes [1997], Carney et al. [1999]. Khosravi et al. [2011] compared four leading techniques for constructing prediction intervals, including the bootstrap. They conclude that there is no method that outperforms all the others in all respects. Tibshirani [1996] compared two different bootstraps to two other methods for estimating the standard error of neural network predictions. Tibshirani [1996] concludes that the bootstraps perform best.

Here we are going to use a method similar to the one proposed by Heskes [1997]. We take bootstrap samples of sequences, rather than individual observations. A separate RNN is trained on each bootstrap sample. Our final (point) prediction is a simple average of all individual model predictions.

We must make a distinction between confidence intervals and prediction intervals. These two are easily confused.

Confidence intervals quantify how well we can approximate the true regression.

The confidence is in the estimate of the regression — the mean of the target distribution.
To compute the intervals, we first construct the bagged estimator by averaging the resulting estimates of all bootstrap runs (13).

$$\hat{y}_i = \frac{1}{n_{run}} \sum_{run=1}^{n_{run}} \hat{y}_i^{run} \tag{13}$$

Now we have the center, and we just need to find the variance (14) and an appropriate value from the t-distribution to calculate the endpoints of the confidence interval (15). We assume that the true regression follows a normal distribution given the estimate.

$$\sigma_{\hat{y}_i}^2 = \frac{1}{n_{run} - 1} \sum_{run=1}^{n_{run}} (\hat{y}_i^{run} - \hat{y}_i)^2 \tag{14}$$

$$CI_i = [\hat{y}_i - t_{conf} \cdot \sigma_{\hat{y}_i},\ \hat{y}_i + t_{conf} \cdot \sigma_{\hat{y}_i}] \tag{15}$$

Prediction intervals quantify how well we can approximate the target values. This measure is more important in practice, but it is a bit more difficult to estimate. While the construction of confidence intervals required nothing more than the means and standard deviations of our resampled estimates, here we need some more sophisticated computations to estimate the noise variance of the regression (16).

$$\sigma_{\hat{\epsilon}}^2 \simeq E[(y - \hat{y})^2] - \sigma_{\hat{y}}^2 \tag{16}$$

We train another network with almost the same structure as the one used for making the time series predictions, and use it to predict the remaining residuals (17) from the input values.

$$r_i^2 = max((y_i - \hat{y}_i)^2 - \sigma_{\hat{y}_i}^2, 0) \tag{17}$$

This residual predictor is trained on the validation set of observations randomly left out in the current bootstrap run. It is a smart data recycling method. Heskes [1997] proposed loglikelihood as the loss function, we apply the similar formula (18) used by Khosravi et al. [2011]. The output activation of this neural network is exponential, so that all predicted error variances are positive.

$$L = \frac{1}{2} \sum_{i=1}^{n} (ln(\sigma_{\hat{\epsilon}_i}^2) + \frac{r_i^2}{\sigma_{\hat{\epsilon}_i}^2}) \tag{18}$$

The output variances and the neural network's predictions are added together (19) to yield estimates of the variance of real interest — the distance of target values from our bagged forecast estimates (20).

$$\sigma_i^2 = \sigma_{\hat{y}_i}^2 + \sigma_{\hat{\epsilon}_i}^2 \tag{19}$$

$$PI_i = [\hat{y}_i - t_{conf} \cdot \sigma_i,\ \hat{y}_i + t_{conf} \cdot \sigma_i] \tag{20}$$

6 Validation

6.1 Point forecasts

We are going to obtain a train and a test set by splitting the time series at one point. It means that we always test the future. We would expect the algorithm to tell the future, so this choice of validation is natural for such forecasting problems. Yet, the test set consists of a single time period, so this method may not be entirely sufficient for evaluating the model performance.
Bootstrapping provides an alternative validation set. An average bootstrap sample contains about 63.2% of the individual observations, or in our case, it contains about 63.2% of all available data subsequences. The remaining subsequences do not participate in the training process, so we may use them for validation purposes.
We are going to call the bootstrap left-out dataset validation set, and the future dataset test set — just for the sake of distinction. They have the same evaluation purpose.
Both sets will be used to evaluate the one-step-ahead forecasting ability of our recurrent neural networks. The separate test set, being a complete chronologically ordered series of subseries, may also be used for iterative multi-step-ahead forecasting.
Regression and classification metrics are going to be applied in order to evaluate the forecasted values and the predicted changes of direction as well.
We are calculating root mean squared error (21), symmetric mean absolute percentage error (22), coefficient of determination (23), mean absolute error (24) and median absolute error (25) regression metrics to measure the forecast fit. We are also calculating the accuracy (MDA) (26), precision (27), recall (28) and F1 (29) classification scores to evaluate the directional forecasts. The classification metrics are computed on the following variables: $\tilde{y}_t = 1_{y_t - y_{t-1} > 0}$ and $\tilde{\hat{y}}_t = 1_{\hat{y}_t - y_{t-1} > 0}$.

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{21}$$

$$SMAPE(y, \hat{y}) = \frac{100}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{([y_i] + |\hat{y}_i|)/2} \tag{22}$$

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{23}$$

$$MAE(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{24}$$

$$MedAE(y, \hat{y}) = median(|y_1 - \hat{y}_1|, \ldots, |y_n - \hat{y}_n|) \tag{25}$$

$$accuracy(\tilde{y}, \tilde{\hat{y}}) = \frac{1}{n} \sum_{i=1}^{n} 1_{\tilde{y}_i = \tilde{\hat{y}}_i} \tag{26}$$

$$precision(\tilde{y}, \tilde{\hat{y}}) = \frac{\sum_{i=1}^{n} 1_{\tilde{y}_i = 1 \; and \; \tilde{\hat{y}}_i = 1}}{\sum_{i=1}^{n} 1_{\tilde{\hat{y}}_i = 1}} \tag{27}$$

$$recall(\tilde{y}, \tilde{\hat{y}}) = \frac{\sum_{i=1}^{n} 1_{\tilde{y}_i=1 \; and \; \tilde{\hat{y}}_i=1}}{\sum_{i=1}^{n} 1_{\tilde{y}_i=1}} \tag{28}$$

$$F1(\tilde{y}, \tilde{\hat{y}}) = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{29}$$

MSE is used as the loss function during the training process. Applying other evaluation metrics is also reasonable, since they all have different properties and interpretation. MAE (the average of absolute errors) is easier to interpret than the square root of MSE, $MedAE$ is more robust to outliers, R^2 score measures the proportion of variance explained by the model, $SMAPE$ measures the error in percentage terms, ranging from 0 to 200%.
Classification *accuracy* score simply measures the proportion of changes whose direction we guessed right. $Precision$, $recall$ and $F1$ are binary metrics. An increase in the target variable is now treated as the positive class. Hence, in our case, *precision* is the proportion of forecasted rises that were right, and *recall* is the proportion of actual rises that we guessed right. $F1$ score is the harmonic mean of precision and recall. We may get a better view of the forecasts' quality by using these various metrics together. Also, we can draw a confusion matrix to evaluate all kinds of errors and correct forecasts at once.

6.2 Interval forecasts

The quality of prediction intervals should also be evaluated. We use the same prediction interval assessment measures as Khosravi et al. [2011].
$PICP$ or prediction interval coverage probability (30) is the proportion of observations that fall into the interval. The wider the interval, the higher the coverage — it is reasonable to quantify the size of the interval as well. $MPIW$ or mean prediction interval width (31) does exactly that. $NMPIW$ or normalized mean prediction interval width (32) is the width normalized by the range of the target variable. This metric allows for comparisons across datasets. CWC or coverage width-based criterion (33) is a combined metric that takes into account both coverage and width. It has 2 hyperparameters that we can set: μ corresponds to the nominal confidence level, while η magnifies the difference between μ and $PICP$. The lower CWC the higher quality.

$$PICP = \frac{1}{n} \sum_{i=1}^{n} 1_{y_i \in [L_i, H_i]} \tag{30}$$

$$MPIW = \frac{1}{n} \sum_{i=1}^{n} (U_i - L_i) \tag{31}$$

$$NMPIW = \frac{MPIW}{R} \tag{32}$$

$$CWC = NMPIW(1 + 1_{PICP<\mu} exp(-\eta(PICP - \mu))) \tag{33}$$

7 Empirical study

Our forecasting framework was implemented in Python (Van Rossum and Drake Jr [1995]). This section presents an example application on the Bike Sharing Dataset (Fanaee-T and Gama [2014]) available in the UCI Machine Learning Repository (Dheeru and Karra Taniskidou [2017]).

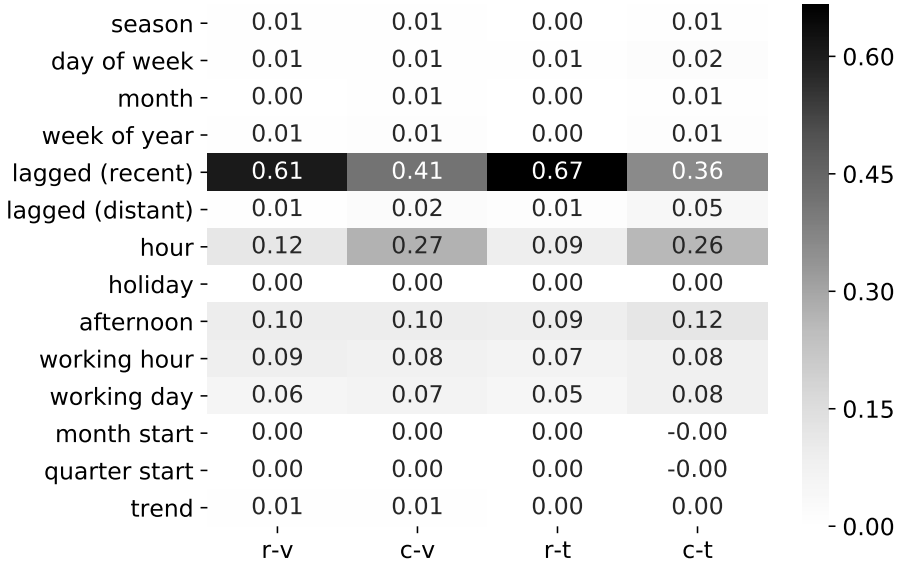


Figure 1: Feature importances, detailed [LSTM]

7.1 Features

The dataset is available in an hourly resolution, which allows us to construct several seasonal variables. It contains counts of bike rentals for Capital Bikeshare at Washington, D.C., USA for 2 years. We disregard the available weather information, and use time-determined features only. The pandas library (McKinney et al. [2010]) was used during the data preparation.

Cyclical features were encoded using the sine and cosine transformations. The following time components are encoded: season of year, month of year, week of year, day of week and hour of day. Using all these features together is probably a bit redundant, but hopefully the neural network is smart enough to handle it. Some binary variables are used to indicate if the time is afternoon, working hour, holiday, working day, start of quarter or start of month. Lagged values of the target variable are also used as inputs: the preceding 3 timesteps' values, and lagged values of 24, 48 and 168 hours.

Each feature was scaled between zero and one using minmax scaler.

Variable importances were calculated on the bootstrap left-out validation set and on the separate test set as well, using the single step forecasts. Two accuracy metrics were applied: R^2 as a measure of goodness-of-fit, and MDA as a measure of directional accuracy. This setting led to 4 different estimates of variable importance (Figures 1 and 2). Those estimates were then averaged into a single list of importances (Figures 3 and 4).

Feature importances are displayed with the following notation: r – regression metric (R^2); c – classification metric (*accuracy*); v – bootstrap left-out validation set; t – future test set. These importance scores were only computed for one-step forecasts, since shuffling the values of the iterative multi-step forecasts was inconvenient.

Lagged values were divided into 2 groups: recent values (lags of 1, 2 and 3

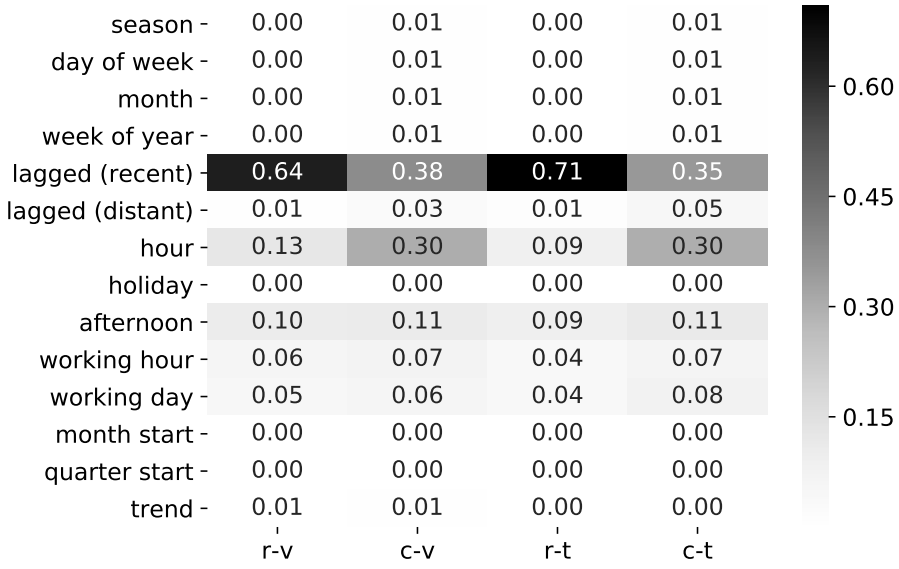


Figure 2: Feature importances, detailed [GRU]

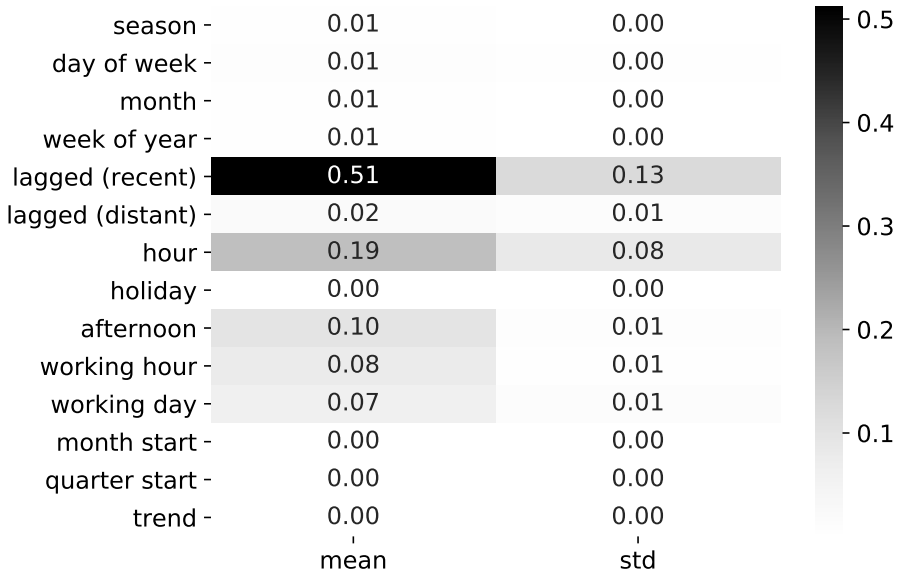


Figure 3: Feature importances, summary [LSTM]

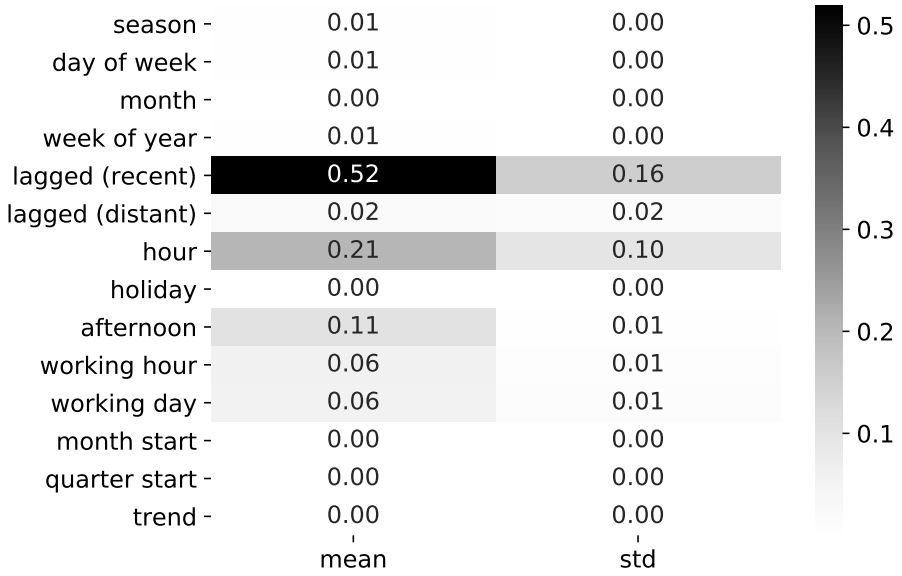
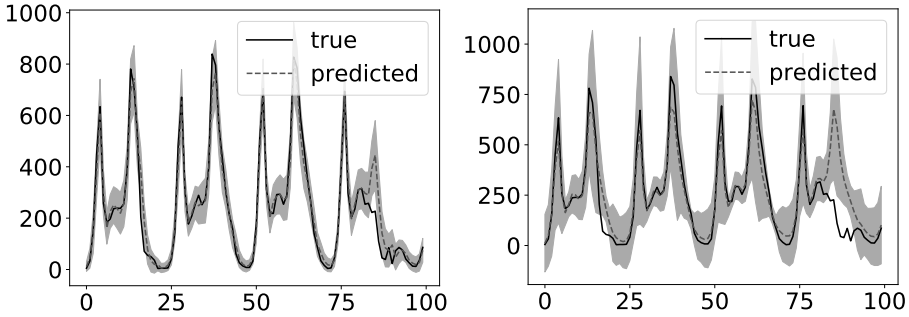


Figure 4: Feature importances, summary [GRU]

hours) and distant values (lags of 24, 48 and 168 hours). Naturally, the sine and cosine transformed values of cyclical features were also treated together. The maximum mean decrease accuracies are reported for these grouped variables.
Recent lagged values constitute the most important group of variables. They are especially important for the value forecasts (regression). Seasonal variables also seem to play an important role — intraday features like the hour of day, afternoon or working hour got large importance scores. The trend variable does not seem to have that much importance, though it was evident during model building attempts that it can add to the RNNs' forecasting capability.
Variable importances were pretty similar for the LSTM and GRU networks. We can hardly see any disagreements in the feature rankings.

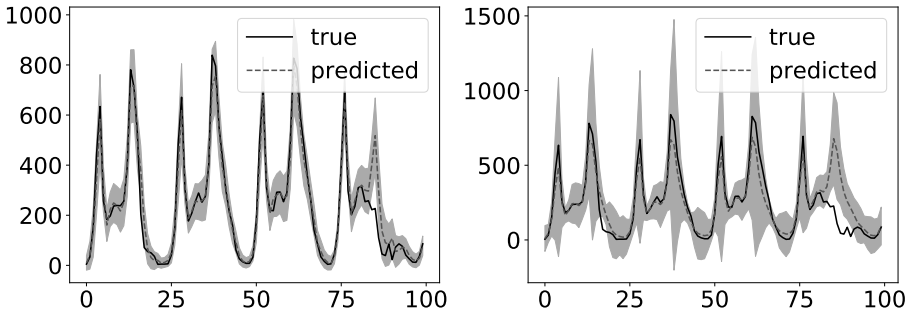
7.2 Predictions

Our recurrent neural networks consist of a one-layer LSTM/GRU of 32 units, followed by a dense layer of a single unit with a linear activation. A dropout of 0.5 was applied to the non-recurrent connections. The learning rate was set to 0.001. The batch size and the number of epochs were 128. The mean squared error loss function was minimized using the Adam optimizer (Kingma and Ba [2014]). 16-step unrolled sequences were fed to the algorithm.
These hyperparameters were not optimized. It is just a reasonable setting for showcasing the method on the given dataset.
The model was trained on 70% of the available data, the last 30% was used as a test set. 50 bootstrap samples were taken from the training set.
Forecasts were made with approximate 90% prediction intervals. Multi-step forecasts were generated for the test set only. Some predictions are shown in Figure 5 and Figure 6.



(a) One-step forecasts (b) Multi-step forecasts

Figure 5: First 100 steps of test set forecasts with 90% prediction intervals [LSTM]



(a) One-step forecasts (b) Multi-step forecasts

Figure 6: First 100 steps of test set forecasts with 90% prediction intervals [GRU]

The recurrent neural networks were built and trained using Keras (Chollet et al. [2015]) with TensorFlow backend (Abadi et al. [2015]).

7.3 Evaluation

Several measures of forecast accuracy and directional accuracy were computed on both evaluation sets.
MSE, $SMAPE$, R^2, MAE and $MedAE$ measure the forecasts' fit, while *accuracy*, $F1$, *precision* and *recall* evaluate the quality of the forecasts' changes of direction. Most of these evaluation metrics were computed using scikit-learn (Pedregosa et al. [2011]).
The results of the regression evaluations are available in Table 1 and Table 2. Table 3 and Table 4 show the change-of-direction binary metrics for the test set. Confusion matrices of the directional predictions are displayed in Figure 7 and Figure 8.
The notations for the tables are the following: b – bagged estimator; i – individual estimators; v – bootstrap left-out validation set; t – future test set; o – one-step forecasts; m – multi-step forecasts.

	$RMSE$	MAE	$SMAPE$	R^2	$MedAE$
v-b-o	31.35	20.25	25.75	0.96	12.84
v-i-o	33.23	21.76	28.81	0.95	13.97
t-b-o	45.74	30.23	22.75	0.96	19.27
t-b-m	90.64	62.99	41.99	0.83	39.56
t-i-o	49.53	33.09	25.19	0.95	21.21
t-i-m	103.31	71.78	44.12	0.78	46.09

Table 1: Regression evaluation metrics [LSTM]

	$RMSE$	MAE	$SMAPE$	R^2	$MedAE$
v-b-o	32.04	20.56	25.63	0.96	12.83
v-i-o	33.94	22.32	29.86	0.95	14.48
t-b-o	48.13	31.24	21.76	0.95	18.61
t-b-m	93.86	64.34	40.32	0.82	36.90
t-i-o	51.97	34.51	26.57	0.94	21.61
t-i-m	105.42	74.49	45.80	0.77	49.07

Table 2: Regression evaluation metrics [GRU]

Multi-step forecasts have consistently higher errors than the one-step-ahead predictions. It is not surprising — just consider the accumulating errors of the iterative forecasting procedure. Lagged values of the target variable proved to be important predictors, especially for forecasting values, rather than just directions. Hence, the accuracy of values that we predict and reuse as inputs, matters a lot. And anyway, the distant future obviously holds much more uncertainty, than the next timestep.

All R^2 values are close to 0.95 for the single-step predictions, and are around 0.8 for the multi-step predictions. The multi-step $RMSE$ is about twice as large as the one-step. The directional accuracies are roughly 85% for the one-step, while around 75% for the multi-step forecasts. For multiple steps, the direction of change forecasts seem to work somewhat better than the value forecasts.

We have averaged the resulting estimates from all bootstrap samples, and used the averages as our final predictions — we used bagging. By comparing the performance of the bagged estimator to the averaged performance of individual estimators, we might evaluate the usefulness of this ensemble method for recurrent neural networks.

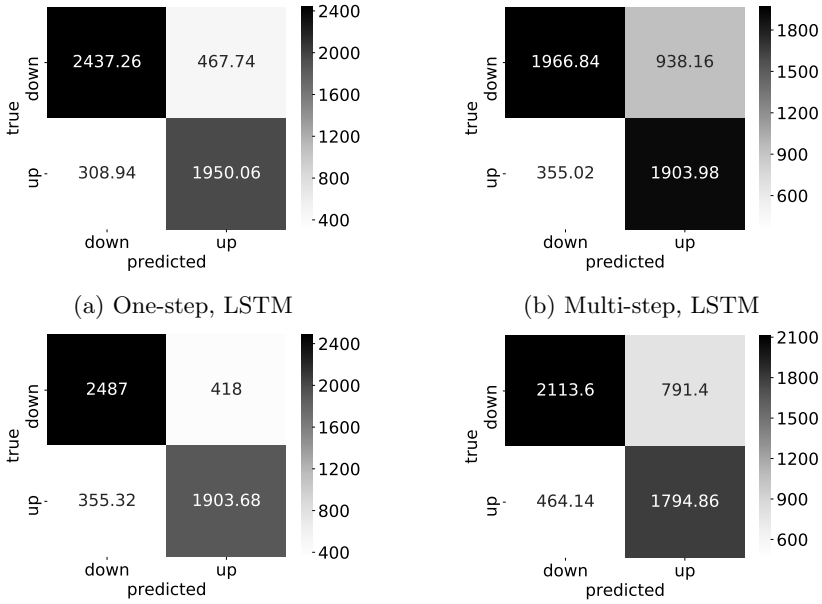
The bagged estimators produced consistently better results than the individual neural networks. This is true for the regression and the classification problem as well. Thus, it seems that bagging can improve RNN-predictions.

	accuracy	precision	recall	F1
b-o	0.87	0.83	0.88	0.85
b-m	0.76	0.68	0.88	0.77
i-o	0.85	0.81	0.86	0.83
i-m	0.75	0.67	0.84	0.75

Table 3: Classification evaluation metrics [LSTM]

	accuracy	precision	recall	F1
b-o	0.87	0.85	0.87	0.86
b-m	0.77	0.70	0.84	0.76
i-o	0.85	0.82	0.84	0.83
i-m	0.76	0.70	0.79	0.74

Table 4: Classification evaluation metrics [GRU]



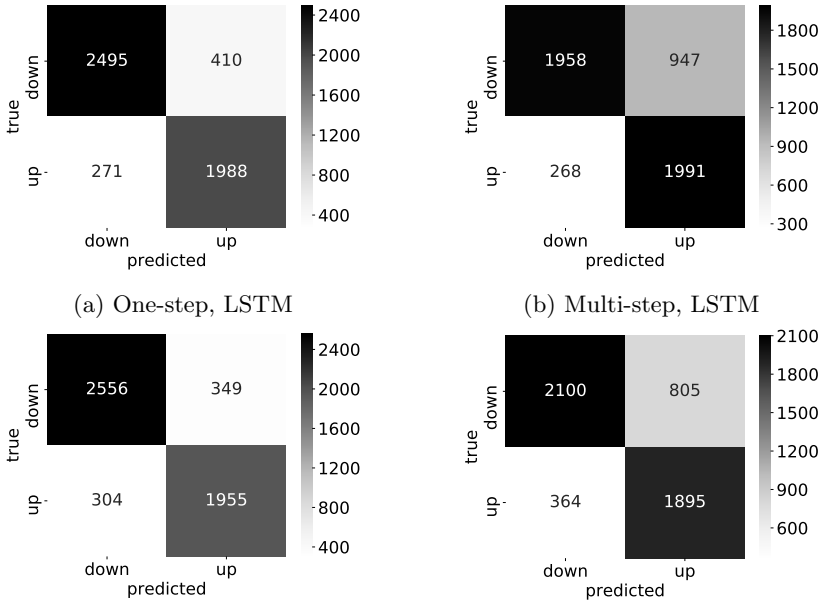
(a) One-step, LSTM

(b) Multi-step, LSTM

(c) One-step, GRU

(d) Multi-step, GRU

Figure 7: Confusion matrices [individual estimates]



(a) One-step, LSTM

(b) Multi-step, LSTM

(c) One-step, GRU

(d) Multi-step, GRU

Figure 8: Confusion matrices [bagged estimates]

The GRU and the LSTM networks are quite close to each other in terms of forecasting performance. The two networks seem to produce very similar forecasts. Papadopoulos et al. [2001] found that the bootstrap consistently overestimates prediction interval coverage. It seems to be confirmed by our results, since all of our prediction intervals have a higher coverage than the targeted 90%. Our residual predictor neural network could probably have been further optimized, in order to generate intervals closer to the desired coverage.
CWC's μ was set to .9, since we aim to generate 90% prediction intervals. η was set to the arbitrary value of 50.
The CWC metrics suggest that the prediction intervals of one-step forecasts are better. It is hardly surprising, since the multi-step forecasts reached similar coverage by producing much wider intervals. The coverage of our prediction intervals exceeds the desired level of 90% in each case, so CWC equals NMPIW. The prediction intervals' evaluation metrics are available in Tables 5 and 6.

	PICP	MPIW	NMPIW	CWC
v-o	0.96	99.91	0.10	0.10
v-m	0.98	313.68	0.33	0.33
t-o	0.94	143.09	0.15	0.15
t-m	0.94	371.85	0.38	0.38

Table 5: PI evaluation metrics [LSTM]

	PICP	MPIW	NMPIW	CWC
v-o	0.96	108.11	0.11	0.11
v-m	0.99	318.90	0.33	0.33
t-o	0.96	161.46	0.17	0.17
t-m	0.95	408.34	0.42	0.42

Table 6: PI evaluation metrics [GRU]

8 Conclusions and Future Perspectives

This study aimed to explore and describe several aspects of the application of recurrent neural networks to time series forecasting, though it is by far not comprehensive.

Recurrent neural networks are much more flexible and much better suited to time series forecasting than the linear models usually applied. Yet, several practices might help their application, some of which have been presented in this article.

We may do time series analysis with the aim of either forecasting future values or understanding the processes driving the time series. Neural networks are particularly bad in the latter. Feature importance measures solve this problem partly. We computed permutation importance scores (mean decrease accuracy). The target variable's lagged values were the most important predictors in our empirical experiment. Seasonality features also seemed important.

Another shortcoming of neural networks is the lack of prediction confidence measures. Interval forecasts can be useful for quantifying uncertainty, though producing them for neural networks is nontrivial. Bootstrapping is a simple, yet computationally expensive method that can do it. However, it produced PIs with consistently higher coverage than what we targeted.

Multiple-step forecasts generated higher errors than single-step forecasts, as expected. The gaps between the errors seemed smaller in case of direction of change predictions.

We found that recurrent neural networks can benefit from bagging.

The LSTM and GRU networks showed about the same forecasting performance. It is hard to argue for either of them.

This forecasting framework might be enhanced in several ways.

During the training process, the states of the LSTM/GRU cells were reset in each batch, so the dependencies between sequences in different batches were not taken into account. We would expect higher accuracies if we took advantage of these relationships in the dataset, though it was difficult with our bootstrapping framework.

Our iterative method is probably not the best solution for making multi-step forecasts. A sequence to sequence learning model (e.g., Cho et al. [2014], Sutskever et al. [2014]) might be a better choice.

We could have constructed further input features. Feature engineering is crucial, and there is always room for improvement. Though neural networks are very flexible, so it didn't seem so necessary.

Feature importances were only computed for one-step forecasts. It would be worth exploring, if different forecasting horizons require different features to make high quality forecasts. Other measures of variable importance could also

be applied.
Feature importances are only a tiny step towards understanding recurrent neural networks. The mechanism of RNN cells could and should be explored in much more depth.
Bootstrapping is computationally intensive, but with today's ever improving GPUs, it is a feasible algorithm for time series datasets of manageable size. Yet, it is a brute force method, so smarter solutions would be welcome.
There are several hyperparameters to optimize — it is also a disadvantage of neural networks. In this article, we did not aim to find the best parameters. Grid search, or rather random search (Bergstra and Bengio [2012]) could have helped in finding the ideal settings.
The (small) size of real world datasets hinders deep learning methods in the field of time series forecasting. If our variable of interest were only observed quarterly or yearly, we would have to wait several lifetimes to acquire a reasonable amount of data. Even this 2-year hourly bike sharing dataset was way too small to exploit the capabilities of a neural network. It would be very useful if we could train an algorithm on multiple similar datasets and gather some collective knowledge that could be used to make better forecasts for the individual time series. This process of gaining knowledge and applying it to solve different problems is called transfer learning. It is already commonly used in, for example, computer vision (Thrun [1996]). Transfer learning is most useful when the training data is scarce, so applying it to time series forecasting seems very promising.
There is so much left to be done. RNNs clearly deserve a seat in the toolbox of time series forecasting.

References

Paul Werbos. Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

Zaiyong Tang, Chrys De Almeida, and Paul A Fishwick. Time series forecasting using neural networks vs. box-jenkins methodology. *Simulation*, 57(5):303–310, 1991.

Julian Faraway and Chris Chatfield. Time series forecasting with neural networks: a comparative study using the air line data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 47(2):231–250, 1998.

G Peter Zhang and Min Qi. Neural network forecasting for seasonal and trend time series. *European journal of operational research*, 160(2):501–514, 2005.

Sandy D Balkin and J Keith Ord. Automatic neural network modeling for univariate time series. *International Journal of Forecasting*, 16(4):509–515, 2000.

Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.

John Cristian Borges Gamboa. Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887*, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

Tian Guo, Zhao Xu, Xin Yao, Haifeng Chen, Karl Aberer, and Koichi Funaya. Robust online time series prediction with recurrent neural networks. In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*, pages 816–825. Ieee, 2016.

Daniel Hsu. Time series forecasting based on augmented long short-term memory. *arXiv preprint arXiv:1707.00666*, 2017.

Yagmur Gizem Cinar, Hamid Mirisaee, Parantapa Goswami, Éric Gaussier, Ali Ait-Bachir, and France Vadim Strijov. Time series forecasting using rnns: an extended attention mechanism to model periods and handle missing values. *CoRR, abs/1703.10089*, 2017.

Kasun Bandara, Christoph Bergmeir, and Slawek Smyl. Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *arXiv preprint arXiv:1710.03222*, 2017.

Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. Time-series extreme event forecasting with neural networks at uber. In *International Conference on Machine Learning*, number 34, pages 1–5, 2017.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Jos van der Westhuizen and Joan Lasenby. Visualizing lstm decisions. *stat*, 1050:23, 2017.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

Chun-Hao Chang, Ladislav Rampasek, and Anna Goldenberg. Dropout feature ranking for deep learning models. *arXiv preprint arXiv:1712.08645*, 2017.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at uber. In *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*, pages 103–110. IEEE, 2017.

Jeffrey A Caley, Nicholas RJ Lawrance, and Geoffrey A Hollinger. Deep networks with confidence bounds for robotic information gathering. 2017.

Muriel Gevrey, Ioannis Dimopoulos, and Sovan Lek. Review and comparison of methods to study the contribution of variables in artificial neural network models. *Ecological modelling*, 160(3):249–264, 2003.

Julian D Olden, Michael K Joy, and Russell G Death. An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological Modelling*, 178(3-4):389–397, 2004.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

B Efron et al. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

Bradley Efron and Robert Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75, 1986.

T Dietterich. Ensemble learning. the handbook of brain theory and neural networks. In ∴ MIT Press, MA, 2002.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Gerhard Paass. Assessing and improving neural network predictions by the bootstrap algorithm. In *Advances in Neural Information Processing Systems*, pages 196–203, 1993.

Tom Heskes. Practical confidence and prediction intervals. In *Advances in neural information processing systems*, pages 176–182, 1997.

John G Carney, Pádraig Cunningham, and Umesh Bhagwan. Confidence and prediction intervals for neural network ensembles. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 2, pages 1215–1218. IEEE, 1999.

Abbas Khosravi, Saeid Nahavandi, Doug Creighton, and Amir F Atiya. Comprehensive review of neural network-based prediction intervals and new advances. *IEEE Transactions on neural networks*, 22(9):1341–1356, 2011.

Robert Tibshirani. A comparison of some error estimates for neural network models. *Neural Computation*, 8(1):152–163, 1996.

Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

Hadi Fanaee-T and Joao Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, 2(2-3):113–127, 2014.

Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

François Chollet et al. Keras. `https://keras.io`, 2015.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Georgios Papadopoulos, Peter J Edwards, and Alan F Murray. Confidence estimation methods for neural networks: A practical comparison. *IEEE transactions on neural networks*, 12(6):1278–1287, 2001.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pages 640–646, 1996.